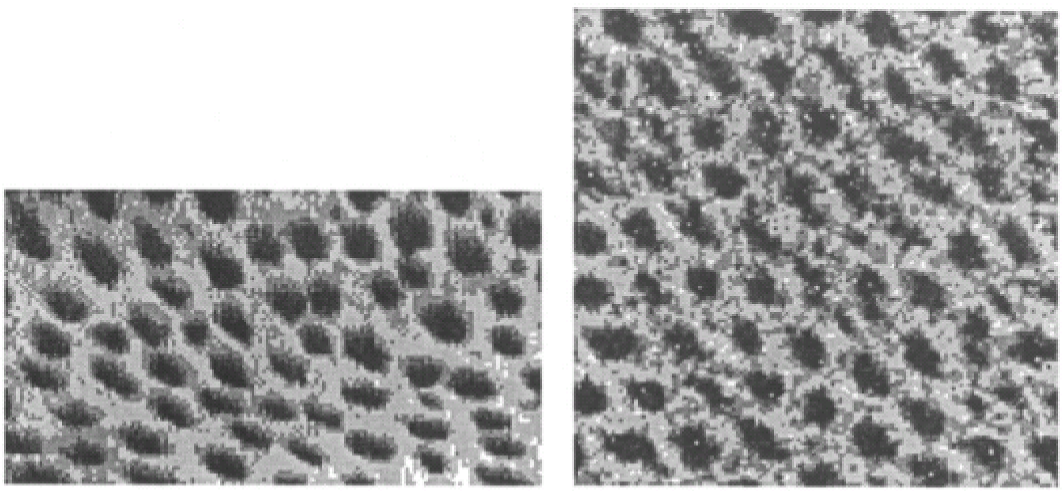
Parallel implementation of FRAME – Filters, Random Fields and Maximum Entropy

Ricardo N. Rodrigues
rnr4@buffalo.edu
CSE633-Parallel Algorithms
University at Buffalo

FRAME Objective

• Song C. Zhu, Yingnian Wu, David Mumford. "Filters, Random Fields and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling", International Journal of Computer Vision, 1998.

- Texture modeling:

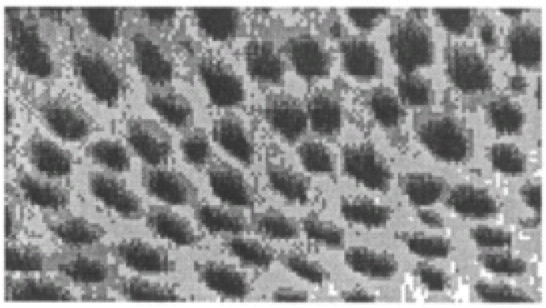


(a) (b)

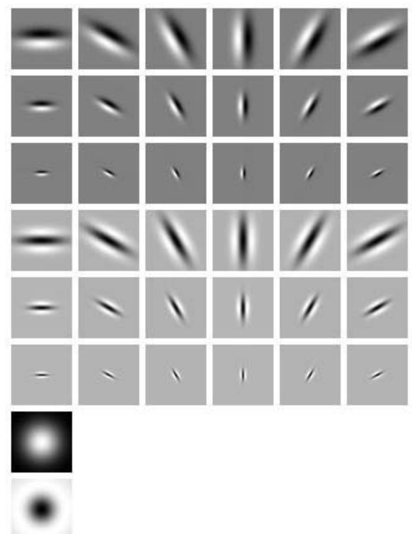
Figure 10. (a) The observed texture—cheetah blob, and (b) the synthesized one using six filters.

Method Overview

Input Image: $\mathbf{I}^{\text{obs}}$

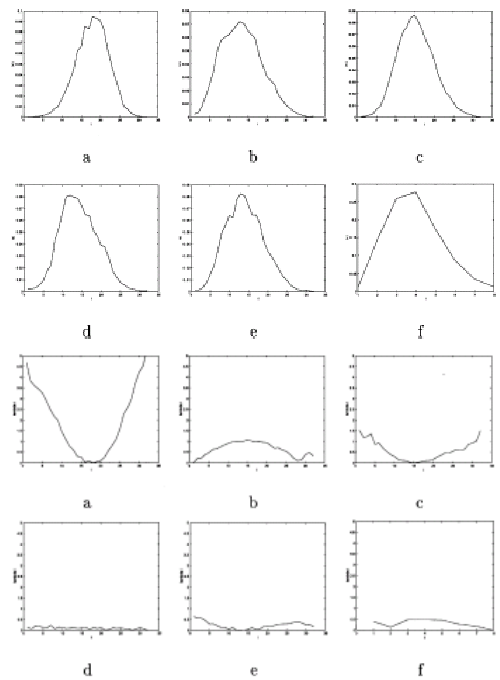


Filters: $S_K = \{F^{(1)}, \ldots, F^{(K)}\}$



Histograms:

$\{H^{\text{obs}(\alpha)}, \quad \alpha = 1, \ldots, K\}.$



Model:

$$p(\mathbf{I}; \Lambda_K, S_K)$$

$$= \frac{1}{Z(\Lambda_K)} \exp\left\{ -\sum_{\alpha=1}^{K} \sum_{i=1}^{L} \lambda_i^{(\alpha)} H_i^{(\alpha)} \right\},$$

$$= \frac{1}{Z(\Lambda_K)} \exp\left\{ -\sum_{\alpha=1}^{K} \langle \lambda^{(\alpha)}, \ H^{(\alpha)} \rangle \right\}.$$

Algorithm

Algorithm 1. The FRAME Algorithm

Input a texture image $\mathbf{I}^{\text{obs}}$.
Select a group of K filters $S_K = \{F^{(1)}, F^{(2)}, \ldots, F^{(K)}\}$.
Compute $\{H^{\text{obs}(\alpha)}, \quad \alpha = 1, \ldots, K\}$.
Initialize $\lambda_i^{(\alpha)} \leftarrow 0, \quad i = 1, 2, \ldots, L, \quad \alpha = 1, 2, \ldots, K$.

Initialize $\mathbf{I}^{\text{syn}}$ as a uniform white noise texture.

Repeat
  Calculate $H^{\text{syn}(\alpha)}$ $\alpha = 1, 2, \ldots, K$ from $\mathbf{I}^{\text{syn}}$, use it for $E_{p(\mathbf{I};\Lambda_K,S_K)}(H^{(\alpha)})$.
  Update $\lambda^{(\alpha)}$ $\alpha = 1, 2, \ldots, K$ by Eq. (19), $p(\mathbf{I};\Lambda_K, S_K)$ is updated.
  Apply Gibbs sampler to flip $\mathbf{I}^{\text{syn}}$ for w *sweeps* under $p(\mathbf{I};\Lambda_K, S_K)$
Until $\frac{1}{2}\sum_i^L |H_i^{\text{obs}(\alpha)} - H_i^{\text{syn}(\alpha)}| \leq \epsilon$ for $\alpha = 1, 2, \ldots, K$.

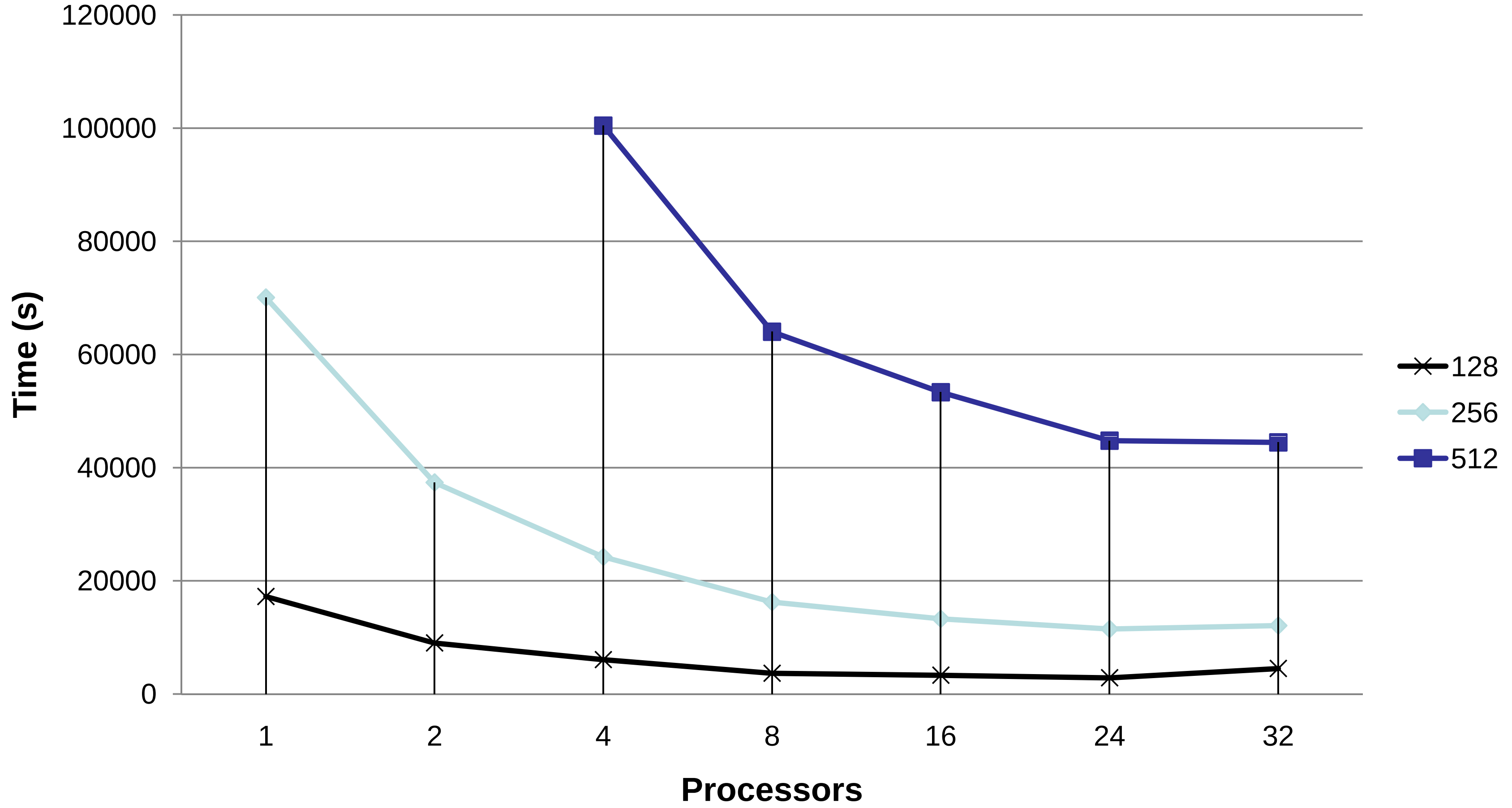
Algorithm 2. The Gibbs Sampler for w Sweeps

Given image $\mathbf{I}(\vec{v})$, flip_counter $\leftarrow 0$
Repeat
  Randomly pick a location $\vec{v}$ under the uniform distribution.
  For val $= 0, \ldots, G-1$ with G being the number of grey levels of $\mathbf{I}$
    Calculate $p(\mathbf{I}(\vec{v}) = \text{val} \mid \mathbf{I}(-\vec{v}))$ by $p(\mathbf{I};\Lambda_K, S_K)$.
  Randomly flip $\mathbf{I}(\vec{v}) \leftarrow \text{val}$ under $p(\text{val} \mid \mathbf{I}(-\vec{v}))$.
  flip_counter $\leftarrow$ flip_counter $+ 1$
Until flip_counter $= w \times M \times N$.

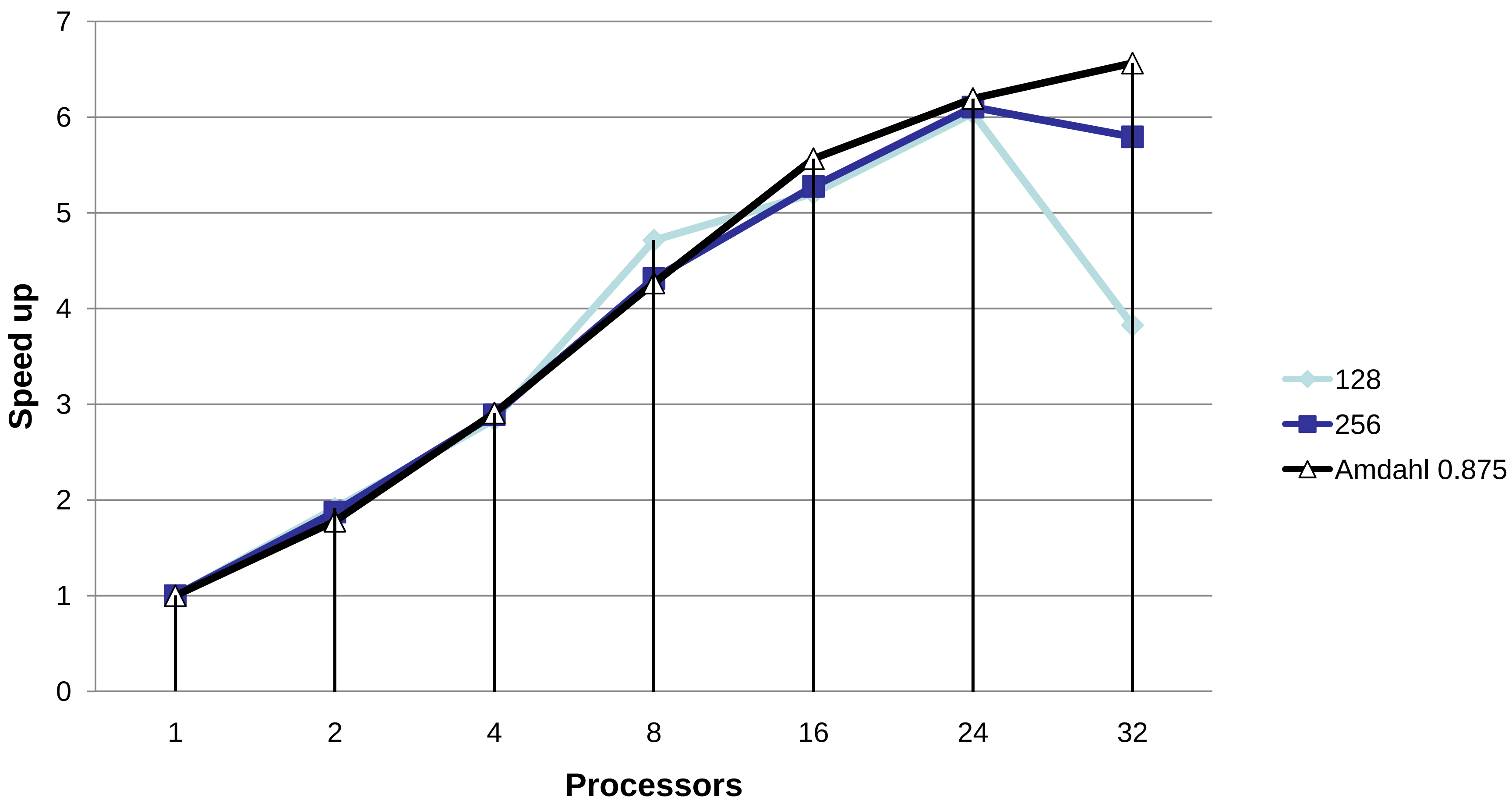
Implementation Issues

- Algorithm 2 (Gibbs sampler) was parallelized.
  - Work shared in the “filter level”
  - So far I have used 24 filters
- Parallelized for shared memory machines using OpenMP
- Parallel region -> 87.5%
- MKL optimizations not running yet due to different versions of MKL in U2 and LENNON.

Running time for different image size



Speed up



7
6
5
4
3
2
1
0
Speed up
1
2
4
8
16
24
32
Processors
128
256
Amdahl 0.875

Serial code profiling using Intel VTune

Sampling Hotspots - [Run 1] | cimg.h | test.cpp | UtilEF.cpp | free.c | MaxEnt.cpp | MaximumEntropy.h | UtilEFT.h

VTune™ Performance Environment

Name	CPU_ sample	INST_R samples	Clocks per...	CPU_CL %	INST_R %	CPU_CLK_UNHA events	INST_RETIRED.A events	Se
get_histogram	419,691	291,960	1.437	37.12%	22.59%	895,200,903,000	622,750,680,000	0xFFF
sampleGivenNatParam	376,855	641,451	0.588	33.33%	49.64%	803,831,715,000	1,368,214,983,000	0xFFF
ftol2_pentium4	119,095	65,102	1.829	10.53%	5.04%	254,029,635,000	138,862,566,000	0xFFF
memcpy	43,575	14,840	2.936	3.85%	1.15%	92,945,475,000	31,653,720,000	0xFFF
ftol2_sse	34,463	108,885	0.317	3.05%	8.43%	73,509,579,000	232,251,705,000	0xFFF
assign<float>	22,149	50,404	0.439	1.96%	3.90%	47,243,817,000	107,511,732,000	0xFFF
mkl_blas_p4m_dgemv	16,675	20,053	0.832	1.47%	1.55%	35,567,775,000	42,773,049,000	0xFFF
fastcopy_I	11,940	11,551	1.034	1.06%	0.89%	25,468,020,000	24,638,283,000	0xFFF
VEC_memzero	8,478	5,878	1.442	0.75%	0.45%	18,083,574,000	12,537,774,000	0xFFF
VEC_memcpy	7,653	3,323	2.303	0.68%	0.26%	16,323,849,000	7,087,959,000	0xFFF

Still working on…

- Extend the use of MKL for more optimization
- Test with different input configurations.